Homework 5 Introduction to Big Data Systems

Christoffer Brevik

October 27, 2024

1 Questions for MapReduce paper

Even though the questions ask me to find a solution to both problems, we actually see that both of these issues are solved the paper on MapReduce. I therefore focus on these solutions when answering the questions.

Q1: Reducing Network Traffic during Shuffle Phase

In the MapReduce framework, the shuffle phase involves transferring the intermediate data generated by the map tasks to the reducers, which can result in significant network traffic. This is especially bad if there is significant repitition in the intermediate keys produced during the map phase. To address this issue, one effective approach is to implement a **combiner function**. The combiner function performs a local, partial aggregation of the intermediate data directly on the mapper nodes. This is very similar to the reduce operation, but it is handled during the map phase and the output is written to an intermediate file that will be sent to a reduce task. By utilizing this, the amount of data transferred over the network can be greatly minimized, thereby reducing network load.

Q2: Addressing Slow Mapper or Reducer Tasks (Stragglers)

While data is typically distributed evenly, failures and bugs in code may cause certain sections to take a longer time to process during both the mapping and reduction phases. These slow tasks, known as *stragglers*, can delay the completion of a MapReduce job. The MapReduce framework mitigates this issue by employing a **backup task** mechanism.

When a job nears completion, the master node initiates duplicate (backup) instances of slow tasks. The job is completed as soon as either the original or any of the backup tasks finishes, after which any duplicate tasks are terminated. This approach helps ensure that the overall job execution is not delayed by any single slow task, thereby improving the completion time of the entire MapReduce operation.

In cases where specific bugs cause certain tasks to fail repeatedly, MapReduce provides an option to **skip problematic records**. If a map or reduce function crashes on a specific record multiple times, the framework detects this repeated failure and skips the problematic record in order to allow the job to make forward progress. This feature is particularly useful when dealing with data inconsistencies or errors that cannot be resolved at runtime, thus improving the reliability and robustness of the MapReduce job even in the presence of repeatable bugs.

2 Implementation of MapReduce

How input data is structured

The `map` function processes each line of input data, which represents edges from a source node to a destination node. The data format, as shown in the ReadMe, is the following:

```
a 1 2 0
a 3 4 0
a 5 1 0
a 2 4 0
a 4 5 0
a 2 5 0
a 2 3 0
a 3 2 0
```

In this format, each line starts with the label `a`, followed by integers representing source and destination nodes. The last number is most likely the weight, and is not used for counting out-degrees.

Updated run_od.sh

I modified the run_od.sh to automatically save the files after running, as well as renaming some variables as these still referred to variables as "words". The updated file is the following:

```
#!/bin/bash

OUTPUTPATH="od_res/"

echo ===== OutDegree JAVA VERSION =====

echo ===== Compile =====
javac -classpath `/hadoop/bin/yarn classpath` OutDegree.java
jar cf od.jar OutDegree*.class
echo

echo ===== Clear old output files on HDFS =====
/hadoop/bin/hdfs dfs -rm -r $OUTPUTPATH
echo

echo ===== RUN CASE1=====
/hadoop/bin/yarn jar od.jar OutDegree /hw5_data/case1 $OUTPUTPATH"case1"
echo

echo ===== RUN CASE2=====
/hadoop/bin/yarn jar od.jar OutDegree /hw5_data/case2 $OUTPUTPATH"case2"
echo

echo ===== Get Result file =====
/hadoop/bin/hdfs dfs -get $OUTPUTPATH .
echo
```

2.1 Explanation of the Map Function

The map function reads each line and uses a `StringTokenizer` to break it into tokens. This is the code in my program:

```
public void map(Object key, Text value, Context context
                        ) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
   itr.nextToken(); // Skip the first token ("a")
            if (itr.hasMoreTokens()) {
                String sourceNode = itr.nextToken(); // The source node
                node.set(sourceNode);
                // Emit the source node with a count of 1 for each outgoing edge
                context.write(node, one);
            }
        }
```

Here's how it processes each line:

- The first token (label) is ignored.
- The function then retrieves the source node and stores it in a `Text` variable.
- For each occurrence of a source node, it emits the source node key with a value of `1`, indicating one outgoing edge.

2.2 Explanation of the Reduce Function

After the Map phase, we run the reduce function. The `reduce` function combines the out-degree counts for each source node based on the output from the map phase. Below is the code used in the program:

```
public void reduce(Text key, Iterable<IntWritable> values,
                           Context context
                        ) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get(); // Sum up all counts for each node
            }
            result.set(sum);
            context.write(key, result); // Emit the node and its total out-degree
        }
```

Here's how the reduce function processes the intermediate data:

- For each unique source node key, the reduce function receives an iterable list of `IntWritable` values, where each value represents one outgoing edge emitted by the map function.
- It initializes a counter, `sum`, to accumulate the out-degree.
- The function iterates over each value in the list and adds it to `sum`.
- After calculating the total count for each source node, the result is set to `sum` and emitted as the final out-degree for that node.

3 Results from running the MapReduce

3.1 Terminal output

Running the `run_od.sh` shell script, we get the following terminal output:

```
===== OutDegree JAVA VERSION =====
===== Compile =====
Note: OutDegree.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

===== Clear old output files on HDFS =====
Deleted od_res

===== RUN CASE1=====
2024-10-27 11:25:36,620 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager
2024-10-27 11:25:37,062 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/h
2024-10-27 11:25:37,340 INFO input.FileInputFormat: Total input files to process : 1
2024-10-27 11:25:37,398 INFO mapreduce.JobSubmitter: number of splits:1
2024-10-27 11:25:37,521 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1727611706614_071
2024-10-27 11:25:37,521 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-10-27 11:25:37,684 INFO conf.Configuration: resource-types.xml not found
2024-10-27 11:25:37,685 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2024-10-27 11:25:37,742 INFO impl.YarnClientImpl: Submitted application application_1727611706614_071
2024-10-27 11:25:37,776 INFO mapreduce.Job: The url to track the job: http://intro00:8088/proxy/appli
2024-10-27 11:25:37,776 INFO mapreduce.Job: Running job: job_1727611706614_0710
2024-10-27 11:25:43,890 INFO mapreduce.Job: Job job_1727611706614_0710 running in uber mode : false
2024-10-27 11:25:43,892 INFO mapreduce.Job:  map 0% reduce 0%
2024-10-27 11:25:47,976 INFO mapreduce.Job:  map 100% reduce 0%
2024-10-27 11:25:53,015 INFO mapreduce.Job:  map 100% reduce 100%
2024-10-27 11:25:53,028 INFO mapreduce.Job: Job job_1727611706614_0710 completed successfully
2024-10-27 11:25:53,193 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=70
                FILE: Number of bytes written=553689
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=986
                HDFS: Number of bytes written=37
                HDFS: Number of read operations=8
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Rack-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=2440
                Total time spent by all reduces in occupied slots (ms)=2300
                Total time spent by all map tasks (ms)=2440
                Total time spent by all reduce tasks (ms)=2300
                Total vcore-milliseconds taken by all map tasks=2440
                Total vcore-milliseconds taken by all reduce tasks=2300
                Total megabyte-milliseconds taken by all map tasks=2498560
                Total megabyte-milliseconds taken by all reduce tasks=2355200
        Map-Reduce Framework
```

```
                Map input records=100
                Map output records=100
                Map output bytes=600
                Map output materialized bytes=70
                Input split bytes=99
                Combine input records=100
                Combine output records=8
                Reduce input groups=8
                Reduce shuffle bytes=70
                Reduce input records=8
                Reduce output records=8
                Spilled Records=16
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=151
                CPU time spent (ms)=1020
                Physical memory (bytes) snapshot=663908352
                Virtual memory (bytes) snapshot=5085540352
                Total committed heap usage (bytes)=1246232576
                Peak Map Physical memory (bytes)=334999552
                Peak Map Virtual memory (bytes)=2539573248
                Peak Reduce Physical memory (bytes)=328908800
                Peak Reduce Virtual memory (bytes)=2545967104
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=887
        File Output Format Counters
                Bytes Written=37

===== RUN CASE2=====
2024-10-27 11:25:55,117 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager
2024-10-27 11:25:55,567 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/h
2024-10-27 11:25:55,846 INFO input.FileInputFormat: Total input files to process : 1
2024-10-27 11:25:55,900 INFO mapreduce.JobSubmitter: number of splits:2
2024-10-27 11:25:56,014 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1727611706614_071
2024-10-27 11:25:56,014 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-10-27 11:25:56,179 INFO conf.Configuration: resource-types.xml not found
2024-10-27 11:25:56,180 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2024-10-27 11:25:56,243 INFO impl.YarnClientImpl: Submitted application application_1727611706614_071
2024-10-27 11:25:56,277 INFO mapreduce.Job: The url to track the job: http://intro00:8088/proxy/appli
2024-10-27 11:25:56,278 INFO mapreduce.Job: Running job: job_1727611706614_0711
2024-10-27 11:26:02,402 INFO mapreduce.Job: Job job_1727611706614_0711 running in uber mode : false
2024-10-27 11:26:02,404 INFO mapreduce.Job:  map 0% reduce 0%
2024-10-27 11:26:12,518 INFO mapreduce.Job:  map 50% reduce 0%
2024-10-27 11:26:18,581 INFO mapreduce.Job:  map 81% reduce 0%
2024-10-27 11:26:24,618 INFO mapreduce.Job:  map 100% reduce 0%
2024-10-27 11:26:25,624 INFO mapreduce.Job:  map 100% reduce 100%
2024-10-27 11:26:25,635 INFO mapreduce.Job: Job job_1727611706614_0711 completed successfully
2024-10-27 11:26:25,768 INFO mapreduce.Job: Counters: 56
```

```
File System Counters
        FILE: Number of bytes read=19623412
        FILE: Number of bytes written=29200919
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=179691884
        HDFS: Number of bytes written=3802485
        HDFS: Number of read operations=11
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
        HDFS: Number of bytes read erasure-coded=0
Job Counters
        Killed map tasks=1
        Launched map tasks=3
        Launched reduce tasks=1
        Data-local map tasks=1
        Rack-local map tasks=2
        Total time spent by all maps in occupied slots (ms)=32400
        Total time spent by all reduces in occupied slots (ms)=10422
        Total time spent by all map tasks (ms)=32400
        Total time spent by all reduce tasks (ms)=10422
        Total vcore-milliseconds taken by all map tasks=32400
        Total vcore-milliseconds taken by all reduce tasks=10422
        Total megabyte-milliseconds taken by all map tasks=33177600
        Total megabyte-milliseconds taken by all reduce tasks=10672128
Map-Reduce Framework
        Map input records=10000000
        Map output records=10000000
        Map output bytes=105233939
        Map output materialized bytes=8747156
        Input split bytes=198
        Combine input records=10847655
        Combine output records=1528498
        Reduce input groups=409565
        Reduce shuffle bytes=8747156
        Reduce input records=680843
        Reduce output records=409565
        Spilled Records=2209341
        Shuffled Maps =2
        Failed Shuffles=0
        Merged Map outputs=2
        GC time elapsed (ms)=661
        CPU time spent (ms)=32240
        Physical memory (bytes) snapshot=1595547648
        Virtual memory (bytes) snapshot=7648034816
        Total committed heap usage (bytes)=2345664512
        Peak Map Physical memory (bytes)=630136832
        Peak Map Virtual memory (bytes)=2552717312
        Peak Reduce Physical memory (bytes)=365457408
        Peak Reduce Virtual memory (bytes)=2550886400
Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
```

```
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=179691686
        File Output Format Counters
                Bytes Written=3802485

===== Get Result file =====
get: ‘od_res/case1/_SUCCESS’: File exists
get: ‘od_res/case1/part-r-00000’: File exists
get: ‘od_res/case2/_SUCCESS’: File exists
get: ‘od_res/case2/part-r-00000’: File exists
```

Here we see that the code has not found any errors when running case1 and case2. Still, we will discuss the result of these two cases run by my algorithm

3.2 Case 1

When the program is run, we can go to `od_res/case1/part-r-00000`. By opening file we get the following:

```
1 31
2 18
3 13
4 10
6 15
7 3
8 8
9 2
```

This is the correct structure as described by the *ReadMe*, where every number is described once and their instances is recorded as a sum to their right.

3.3 Case 2

We can also open `od_res/case2/part-r-00000`. This file has 409565 lines and therefore unsuitable to display in this report. I will, however, show the first and last 5 entries:

```
1      14668
100      372
1000 363
100001 6
100002 1
...
99994 12
999941 1
99995 4
99998 18
99999 5
```

This file can also be found in the code also found in the homework.